IEEE COMPUTER SOCIETY

EXECUTIVE BRIEFING

DOCUMENT IMAGE ANALYSIS

Lawrence O'Gorman
Rangachar Kasturi

IEEE COMPUTER SOCIETY IEEE

Executive Briefing

Document Image Analysis

Lawrence O'Gorman
Rangachar Kasturi

IEEE
COMPUTER
SOCIETY

Los Alamitos, California

Washington • Brussels • Tokyo

Library of Congress Cataloging-in-Publication Data

O'Gorman, Lawrence.
Executive briefing: document image analysis / Lawrence O'Gorman, Rangachar Kasturi.
p. cm.
Includes bibliographical references.
ISBN 0-8186-7802-X
1. Document imaging systems. 2. Image processing—Digital techniques. 3. Optical character recognition devices. I. Kasturi, Rangachar, 1949– . II. Title.
HF5737.037 1997
006.4 ' 2—dc21

97-17283
CIP

IEEE Computer Society Press Order Number BR07802
Library of Congress Number 97-17283
ISBN 0-8186-7802-X

Publisher: Matt Loeb
Acquisitions Editor: Bill Sanders
Developmental Editor: Cheryl Smith
Advertising/Promotions: Tom Fink
Production Editor: Lisa O'Conner
Printed in the United States of America by KNI, Incorporated

IEEE COMPUTER SOCIETY

IEEE

Preface

In the late 1980s, the prevalence of fast computers, large computer memory, and inexpensive scanners fostered an increasing interest in document image analysis. With many paper documents being sent and received via fax machines and stored digitally in large document databases, interest grew in doing more than simply viewing and printing these images. Research was undertaken and commercial systems built to read text on a page, to find fields on a form, and to locate lines and symbols on a diagram. Today, we see the results of this research and development in document processing and optical character recognition (OCR). OCR is used by post offices to automatically route mail; engineering diagrams are extracted from paper for computer storage and modification; handheld computers recognize symbols and handwriting for use in niche markets such as inventory control. In the future, applications such as these will be improved, and other document applications will be added. For instance, the millions of paper volumes now in libraries will be replaced by computer files of page images that can be searched for by content and accessed by many people simultaneously—and they will never be misshelved. Businesspeople will carry their file cabinets in their portable computers; paper copies of new product literature, receipts, or other random notes will be instantly filed and accessed in the computer; and signatures will be analyzed by the computer for verification and security access.

This book describes some of the technical methods and systems used for document processing of text and graphics images. The methods have grown out of the fields of digital signal processing, digital image processing, and pattern recognition. The objective is to give the reader an understanding of what approaches are used for application to documents and how these methods apply to different situations. Since the field of document processing is relatively new, it is also dynamic; in other words, current methods have room for improvement, and innovations are still being made. In addition, there are rarely definitive techniques for all cases of a certain problem.

The intended audience is executives, managers, and other decision makers whose businesses require some acquaintance or understanding of document processing. (We call this group "executives" in accordance with the *Executive Briefing* series.) Some rudimentary knowledge of computers and computer images will be helpful background for these readers. We begin with basic principles (such as defining pixels) but do not belabor them. The reader is expected to have a level of comfort with the tasks that can be accomplished on a computer and the digital nature by which any computer technique operates and not necessarily knowledge of picture processing. A grasp of the terminology goes a long way toward aiding the executive in understanding the technology and processes discussed in each chapter. For this reason, each section begins with a list of keywords. With knowledge of the terminology and whatever depth of method or system understanding that he or she decides to take from the text, the executive should be well equipped to deal with document-processing issues.

In each chapter, we attempt to identify major problem areas and to describe more than one method applied to each problem, as well as the advantages and disadvantages of each method. This gives an understanding of the problems and also the nature of trade-offs that so often must be made in choosing a method. With this understanding of the problem and a knowledge of the methodology options, an executive will have the technical background and context with which to ask questions, judge recommendations, weigh options, and make decisions.

We include technology descriptions and references to the technical papers that best give details on the techniques presented in the book. The technology descriptions are detailed enough for one to understand the methods—if implementation is desired, the references will facilitate this. Popular as well as accepted methods are presented so that the executive can compare a variety of options. In many cases, some of the options are advanced methods not currently used in commercial products. Depending on the level of need, advanced methods can be implemented by the programming staff to yield better results. We also describe many full systems entailing document processing. These are described from the applications' viewpoint to give concrete examples of where and how the methods are implemented.

The book is organized in the sequence that document images are usually processed. After document input by digital scanning, pixel processing is first performed. This level of processing includes operations that are applied to all image pixels. These include noise removal, image enhancement, and segmentation of image components into text and graphics (lines and symbols). Feature-level analysis treats groups of pixels as entities and includes line and curve detection and shape description. The last

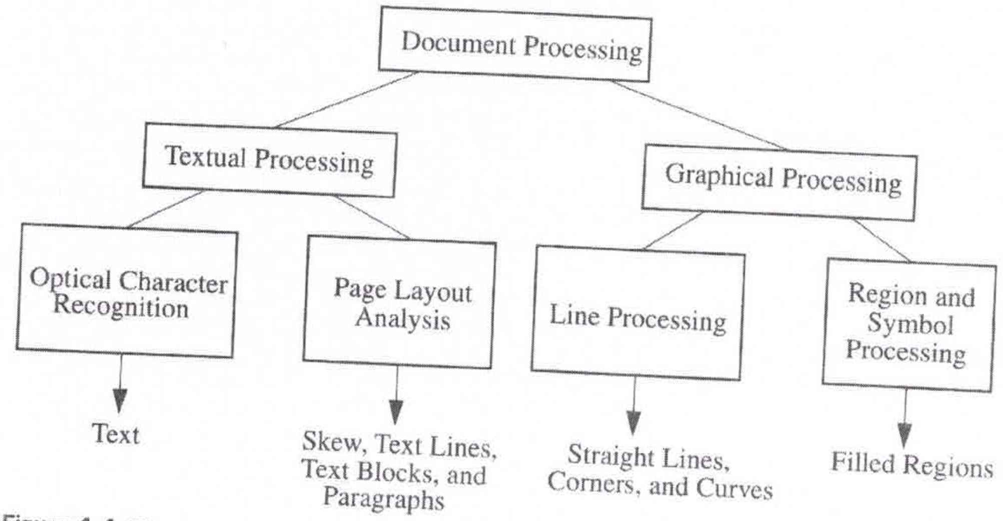

Document Processing
Textual Processing
Graphical Processing
Optical Character Recognition
Page Layout Analysis
Line Processing
Region and Symbol Processing
Text
Skew, Text Lines, Text Blocks, and Paragraphs
Straight Lines, Corners, and Curves
Filled Regions

Figure 1.1 Hierarchy of document processing subareas listing the types of document components in each subarea.

sist of lines, such processing includes line thinning, line fitting, and corner and curve detection. (The use of the word *line* in this book means straight, curved, or piecewise straight and/or curved lines. When straightness or curvedness is important, it will be specified.) Pictures are another major component of image analysis; however, except for recognizing the location of pictures on a page, further analysis of pictures is usually the task of other image processing and machine vision techniques, so we do not deal with picture processing in this book. After application of these text and graphics analysis techniques, the megabytes of initial data are culled to yield a concise semantic description of the document.

It is not difficult to find examples of the need for document analysis. Look around the workplace and you will see stacks of paper documents. Some may be computer generated, though invariably by different computers and software, and their electronic formats may be incompatible. Some will include both formatted text and tables, as well as handwritten entries, and they differ in size, from 3.5 × 2 in. (8.89 × 5.08cm) business cards to 34 × 44 in. (86 × 111cm) engineering drawings. In many businesses today, imaging systems are used to store images of pages so that storage and retrieval are more efficient. Future document analysis systems will recognize types of documents, enable the extraction of their functional parts, and be able to translate from one computer generated format to another. Many other examples exist of the use of and need for document systems. Glance behind the counter in a post office at the mounds of letters and packages. In some U.S. post offices, over a million pieces of mail must be handled each day. Machines to perform sorting and address recognition have been used for several decades, but there is still the need to process more mail, more quickly, and more accurately. Examine the stacks of a library, where row after row of paper documents are stored. Loss of material, misfiling, limited numbers of each copy, and even degradation of materials are common problems and may be improved by document analysis techniques. All of these examples serve as applications ripe for the potential solutions of document image analysis.

Though document image analysis has been in use for a couple of decades (especially in the banking business for computer reading of numeric check codes), only in the late 1980s and early 1990s has the field grown rapidly. The predominant reason for this is the greater speed and lower cost of hardware now available. Since fax machines have become ubiquitous, the cost of optical scanners for document input has dropped to the level that these are affordable to even small businesses and individuals. Although document images contain a relatively large amount of data, even personal computers now have adequate speed to process them. Computer main memory also is now adequate for large document images; more importantly, however, optical memory is now available for mass storage of large amounts of data. This improvement in hardware, and the increased use of computers for storing paper documents, has led to increasing interest in improving the technology of document processing and recognition. The advancements being made in document analysis software and algorithms are an essential complement to these hardware improvements. With OCR recognition rates now in the mid to high 90 percent range, and other document processing methods achieving similar improvements, these advances in research have also driven document image analysis forward.

As improvements in technology continue, document systems will become increasingly more common. For instance, OCR systems will be more widely used to store, search, and excerpt from paper-based documents. Page layout analysis techniques will recognize a particular form or page format and allow its duplication. Diagrams will be entered from pictures or by hand and logically edited. Pen-based computers will translate handwritten entries into electronic documents. Archives of paper documents in libraries and engineering companies will be electronically converted for more efficient storage and instant delivery to a home or office computer. Although it will be increasingly the case that documents are produced and reside on a computer, because there are many different systems and protocols and because paper is a very comfortable medium for us to deal with, paper documents will be with us to some degree for many decades to come. The difference will be that they will finally be integrated into our computerized world.

1.1 Hardware Advancements and the Evolution of Document Image Analysis

The history of document image analysis can be traced through a computer lineage that includes digital signal processing and digital image processing. Digital signal processing, whose study and use was initially fostered by the introduction of fast computers and algorithms, such as the fast Fourier transform in the mid-1960s, has as its objective the interpretation of one-dimensional signals, such as speech and other audio. In the early 1970s, with larger computer memories and still faster processors, image processing methods and systems were developed for analysis of two-dimensional signals, including digitized pictures. Special fields of image processing are associated with particular applications—for example, biomedical image processing for medical images, machine vision for processing of pictures in industrial manufacturing, and computer vision for processing images of three-dimensional scenes used in robotics.

In the mid- to late-1980s, document image analysis began to grow rapidly. Again, this was predominantly due to hardware advancements enabling processing to be performed at a reasonable cost and speed. Whereas a speech signal is typically processed in frames of 256 samples long and a machine vision image size of 512 × 512 pixels, a document image is from 2,550 × 3,300 pixels for a business letter digitized at 300 dots per inch (dpi) (12 dots per millimeter) to 34,000 × 44,000 pixels for a 34 × 44 in. E-sized engineering diagram digitized at 1,000 dpi. Commercial document analysis systems are now available for storing business forms, performing OCR on typewritten text, and compressing engineering drawings. Document analysis research continues to pursue more intelligent handling of documents, better compression, especially through component recognition, and faster processing.

1.2 From Pixels to Paragraphs and Drawings

Figure 1.2 illustrates a common sequence of steps in document image analysis. This is also the organization of chapters in this book. After data capture, the image undergoes pixel-level processing and feature analysis, then text and graphics are treated separately for recognition of each.

Data capture is performed on a paper document, usually by optical scanning. The resulting data is stored in a file of picture elements, called pixels, that are sampled in a grid pattern throughout the document. These pixels may have values: OFF (0) or ON (1) for binary images, 0 to 255 for gray-scale images, and three channels of 0 to 255 color values for color images. At a typical sampling resolution of 300 dpi, a 8.5 × 11 in. page would yield an image of 2,550 × 3,300 pixels. It is important to understand that the image of the document contains only raw data that must be further analyzed to glean the information. For example, Figure 1.3 shows the image of the letter *e*. This is a pixel array of ON or OFF values whose shape is known to humans as the letter *e*; however, to a computer it is just a string of bits in computer memory.

1.2.1 Pixel-Level Processing (Chapter 2)

This stage of document image analysis includes binarization, noise reduction, signal enhancement, and segmentation. For gray-scale images with information that is inher-

7,500

1,500 Wo
1 Title, 2

Figure 1.2

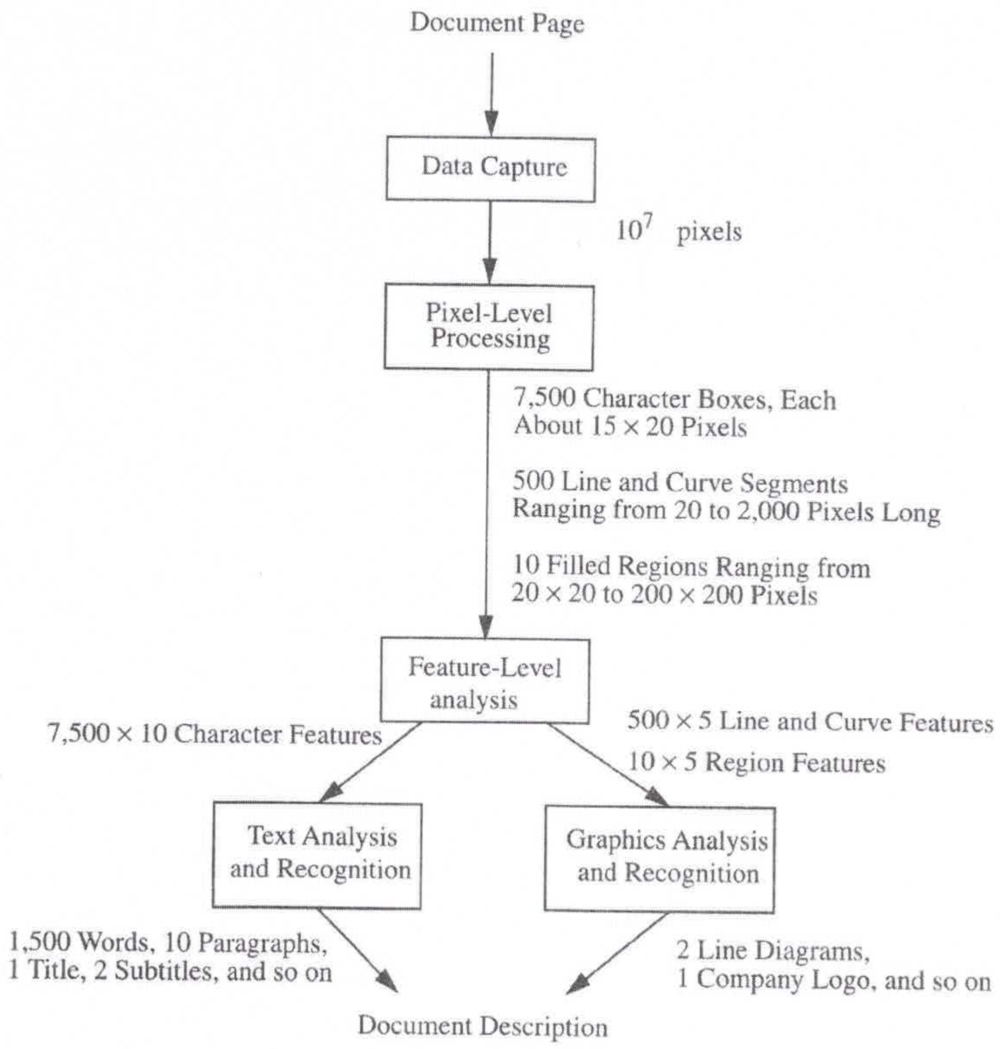

Document Page
Data Capture
10^7 pixels
Pixel-Level Processing
7,500 Character Boxes, Each About 15 × 20 Pixels
500 Line and Curve Segments Ranging from 20 to 2,000 Pixels Long
10 Filled Regions Ranging from 20 × 20 to 200 × 200 Pixels
Feature-Level analysis
7,500 × 10 Character Features
500 × 5 Line and Curve Features
10 × 5 Region Features
Text Analysis and Recognition
Graphics Analysis and Recognition
1,500 Words, 10 Paragraphs, 1 Title, 2 Subtitles, and so on
2 Line Diagrams, 1 Company Logo, and so on
Document Description

Figure 1.2 A typical sequence of steps for document analysis, with examples of intermediate and final results and data size.

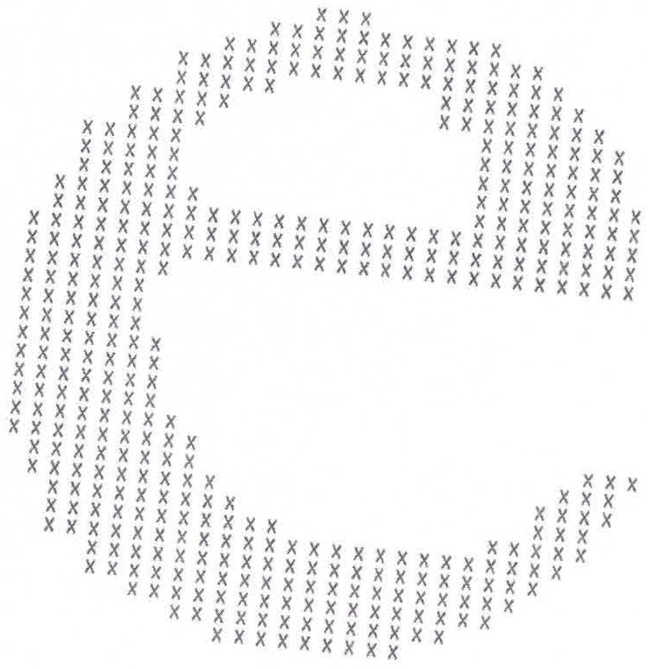


Figure 1.3 Binary image of the letter *e* using ON and OFF pixels; the ON pixels are shown here as Xs.

ently binary, such as text or graphics, binarization is usually performed first. The objective in methods for binarization is to automatically choose a threshold that separates the foreground and background information. An example that has seen much research is the extraction of handwriting from a bank check—especially a bank check containing a scenic background image.

Document image noise occurs from image transmission, photocopying, or degradation due to aging. Salt-and-pepper noise (also called impulse noise, speckle noise, or just dirt) is a common form of noise on a binary image. It consists of randomly distributed black specks on a white background and white specks on a black background. Noise reduction is accomplished by performing filtering on the image so that the background is "grown" into the noise specks, thus filling these holes. Signal enhancement is similar to noise reduction but uses domain knowledge to reconstitute expected parts of the signal that have been lost. Signal enhancement is often applied to graphics components of document images to fill gaps in lines that are known to be continuous otherwise.

Segmentation occurs on two levels. On the first level, segmentation occurs if the document contains both text and graphics—these are separated for subsequent processing by different methods. On the second level, segmentation is performed on text by locating columns, paragraphs, words, titles, and captions; and on graphics by separating symbol and line components. For instance, in a page containing a flow chart with an accompanying caption, text and graphics are first separated. Then the text is separated into that of the caption and that of the chart. The graphics are separated into rectangles, circles, connecting lines, and so on.

1.2.2 Feature-Level Analysis (Chapter 3)

In a text image, the global features describe each page and consist of skew (the tilt at which the page has been scanned), line lengths, line spacing, and so on. Local features describe individual characters and consist of font size, number of loops in a character, number of crossings, accompanying dots and so on.

In a graphical image, global features describe the skew of the page, the line widths, range of curvature, minimum line lengths, and so on. Local features describe each corner, curve, and straight line, as well as the rectangles, circles, and other geometric shapes.

1.2.3 Recognition of Text and Graphics (Chapters 4 and 5)

The final step in document image analysis is recognition and description: components are assigned a semantic label and the entire document is described as a whole. Domain knowledge is used most extensively at this stage. The result is a description of a document as a human would give it. For a text image, we refer for example, not to pixel groups or blobs of black on white, but to titles, subtitles, bodies of text, and footnotes. Depending on the arrangement of these text blocks, a page of text may be a title page of a paper, a table of contents of a journal, a business form, or the face of a mail piece. For a graphical image, an electrical circuit diagram for instance, we refer not to lines joining circles and triangles and other shapes, but to connections between AND gates, transistors, and other electronic components. The components and their connections describe a particular circuit that has a purpose in the known domain. It is this semantic description that is most efficiently stored and most effectively used for common tasks, such as indexing and modifying particular document components .

Chapter 2

Preparing the Document Image

2.1 Introduction

Data capture of documents by optical scanning or by digital video yields a file of picture elements, or pixels, that is the raw input to document analysis. These pixels are samples of intensity values taken in a grid pattern over the document page, where the intensity values may be OFF (0) or ON (1) for binary images, 0 to 255 for gray-scale images, and three channels of 0 to 255 color values for color images. The first step in document analysis is to perform processing on this image to prepare it for further analysis. Such processing includes thresholding to reduce a gray-scale or color image to a binary image, reduction of noise to reduce extraneous data, and thinning and region detection to enable easier subsequent detection of pertinent features and objects of interest. This pixel-level processing (also called preprocessing and low-level processing in other literature) is the subject of this chapter.

2.2 Thresholding

Keywords: thresholding, binarization, global thresholding, adaptive thresholding, intensity histogram

In this treatment of document processing, we deal with images containing text and graphics of binary information—that is, these images contain a single foreground level that is the text and graphics of interest and a single background level upon which the foreground contrasts. We will also call the foreground objects, regions of interest, or components. (Of course, documents may also contain true gray-scale [or color] information, such as in photographic figures; however, except for recognizing the presence of a gray-scale picture in a document, we leave the analysis of pictures to the more general fields of image analysis and machine vision.) Although the information is binary, the data—in the form of pixels with intensity values—are not likely to have only two levels, they, instead, have a range of intensities. This may be due to non-uniform printing or non-uniform reflectance from the page or a result of intensity transitions at the region edges that are located between foreground and background regions. The objective in binarization is to mark pixels that belong to true foreground regions with a single intensity (ON) and background regions with a different intensity (OFF). Figure 2.1 illustrates the results of binarizing a document image at different threshold values. The ON values are shown in black in Figure 2.1, and the OFF values are in white.

For documents with a good contrast of components against a uniform background, binary scanners are available that combine digitization with thresholding to yield binary data; however, for the many documents that have a wide range of background and object intensities, this fixed threshold level often does not yield images with clear separation between the foreground components and the background. For instance, when a document is printed on differently colored paper, when the foreground components are faded due to photocopying, or when different scanners have different light levels, the best threshold value will also be different. For these cases, there are two alternatives. One is to empirically determine the best binarization setting on the scanner (most binary scanners provide this adjustment) and to do this each time an image is poorly binarized. The other alternative is to start with gray-scale images (having a range of intensities, usually from 0 to 255) from the digitization stage and then to use methods for automatic threshold determination to better perform binarization. Although the latter alternative requires more input data and processing, the advantage is that a good threshold level can be found automatically, ensuring consistently good images, and precluding the need for time-consuming manual adjustment and repeated digitization. The following discussion presumes initial digitization to gray-scale images.

If the pixel values of the components and those of the background are fairly consistent in their respective values over the entire image, a single threshold value can be found for the image. This use of a single threshold for all image pixels is called global thresholding. Processing methods will be described that automatically determine the best global threshold value for different images. For many documents, however, a single global threshold value cannot be used even for a single image due to non-uniformities within foreground and background regions. For example, for a document containing white background areas as well as highlighted areas of a different background color, the best thresholds will change by area. For this type of image, different threshold values are required for different local areas; this is adaptive thresholding and will also be described.

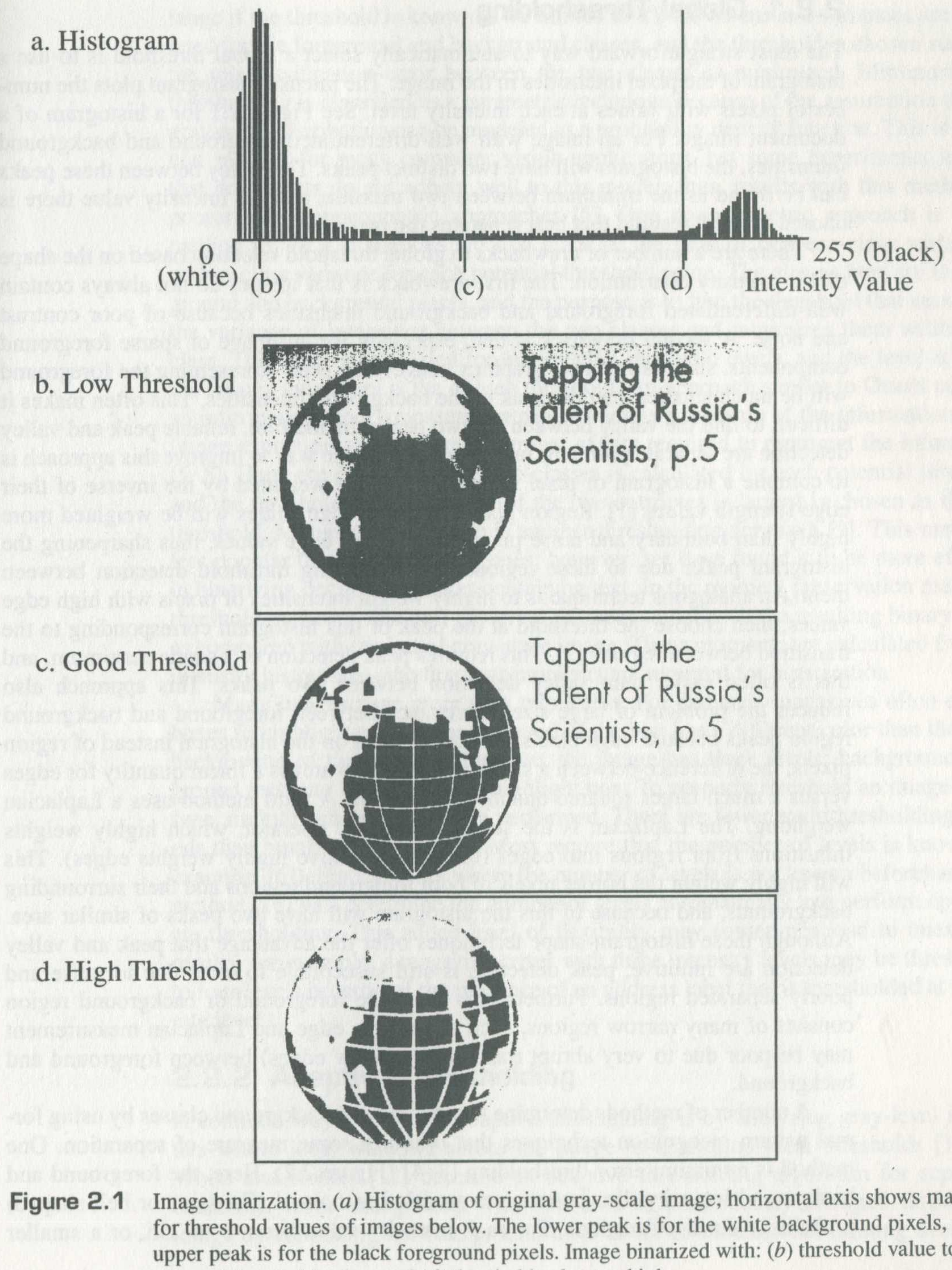


Figure 2.1 Image binarization. (*a*) Histogram of original gray-scale image; horizontal axis shows markings for threshold values of images below. The lower peak is for the white background pixels, and the upper peak is for the black foreground pixels. Image binarized with: (*b*) threshold value too low, (*c*) good threshold value, and (*d*) threshold value too high.

2.2.1 Global Thresholding

The most straightforward way to automatically select a global threshold is to use a histogram of the pixel intensities in the image. The intensity histogram plots the number of pixels with values at each intensity level. See Figure 2.1 for a histogram of a document image. For an image with well-differentiated foreground and background intensities, the histogram will have two distinct peaks. The valley between these peaks can be found as the minimum between two maxima, and the intensity value there is chosen as the threshold that best separates the two peaks.

There are a number of drawbacks to global threshold selection based on the shape of the intensity distribution. The first drawback is that images do not always contain well-differentiated foreground and background intensities because of poor contrast and noise. A second drawback is that, especially for an image of sparse foreground components, such as for most graphics images, the peak representing the foreground will be much smaller than the peak of the background intensities. This often makes it difficult to find the valley between the two peaks. In addition, reliable peak and valley detection are separate problems unto themselves. One way to improve this approach is to compile a histogram of pixel intensities that are weighted by the inverse of their edge-strength values [1]. Region pixels with low edge values will be weighted more highly than boundary and noise pixels with higher edge values, thus sharpening the histogram peaks due to these regions and facilitating threshold detection between them. An analogous technique is to highly weight intensities of pixels with high edge values, then choose the threshold at the peak of this histogram corresponding to the transition between regions [2]. This requires peak detection of a single maximum, and this is often easier than valley detection between two peaks. This approach also reduces the problem of large size discrepancy between foreground and background region peaks because edge pixels are accumulated on the histogram instead of region pixels; the difference between a small and large size area is a linear quantity for edges versus a much larger squared quantity for regions. A third method uses a Laplacian weighting. The Laplacian is the second derivative operator, which highly weights transitions from regions into edges (the first derivative highly weights edges). This will highly weight the border pixels of both foreground regions and their surrounding backgrounds, and because of this the histogram will have two peaks of similar area. Although these histogram-shape techniques offer the advantage that peak and valley detection are intuitive, peak detection is still susceptible to error due to noise and poorly separated regions. Furthermore, when the foreground or background region consists of many narrow regions, such as for text, edge and Laplacian measurement may be poor due to very abrupt transitions (narrow edges) between foreground and background.

A number of methods determine foreground and background classes by using formal pattern recognition techniques that optimize some measure of separation. One method is minimum error thresholding [3, 4] (Figure 2.2). Here, the foreground and background intensity distributions are modeled as normal (Gaussian or bell-shaped) probability density functions. For each intensity value (from 0 to 255, or a smaller range if the threshold is known to be limited to it), the means and variances are calculated for the foreground and background classes, and the threshold is chosen such that the misclassification error between the two classes is minimized. Minimum error thresholding is classified as a parametric technique because of the assumption that the gray-scale distribution can be modeled as a probability density function. This is a popular method for many computer vision applications, but some experiments indicate that documents do not adhere well to this model; thus, results with this method are poorer than nonparametric approaches [5]. One nonparametric approach is Otsu's method [6, 7]. Calculations are first made of the ratio of between-class variance to within-class variance for each potential threshold value. The classes here are the foreground and background pixels, and the purpose is to find the threshold that maximizes the variance of intensities between the two classes and minimizes them within each class. This ratio is calculated for all potential threshold levels, and the level at which the ratio is maximum is the chosen threshold. An approach similar to Otsu's employs an information theory measure, entropy, which is a measure of the information in the image expressed as the average number of bits required to represent the information [5, 8]. Here, the entropy for the two classes is calculated for each potential threshold, and the threshold where the sum of the two entropies is largest is chosen as the best threshold. Moment preservation is another thresholding approach [9]. This method is less popular than the preceding ones; however, we have found it to be more effective in binarizing document images containing text. In the moment preservation method, a threshold is chosen that best preserves moment statistics in the resulting binary image as compared with the initial gray-scale image. These moments are calculated from the intensity histogram—the first four moments are required for binarization.

Many images have more than two levels. For instance, magazines often employ boxes to highlight text; the background of the box has a different color than the white background of the page. In this case, the image has three levels: background, foreground text, and background of highlight box. To properly threshold an image of this type, multithresholding must be performed. There are fewer multithresholding methods than binarization methods. Most require that the number of levels is known (for example, [6]). For the cases where the number of levels is not known beforehand, one method [16] will determine the number of levels automatically and perform appropriate thresholding. This added level of flexibility may sometimes lead to unexpected results; for example, a magazine cover with three intensity levels may be thresholded to four levels because of the presence of an address label that is thresholded at a separate level.

2.2.2 Adaptive Thresholding

A common way to perform adaptive thresholding is by analyzing gray-level intensities within local windows across the image to determine local thresholds [10, 11]. White and Rohrer [12] describe an adaptive thresholding algorithm for separating characters from background. The threshold is continuously changed through the image by estimating the background level as a two-dimensional running average of

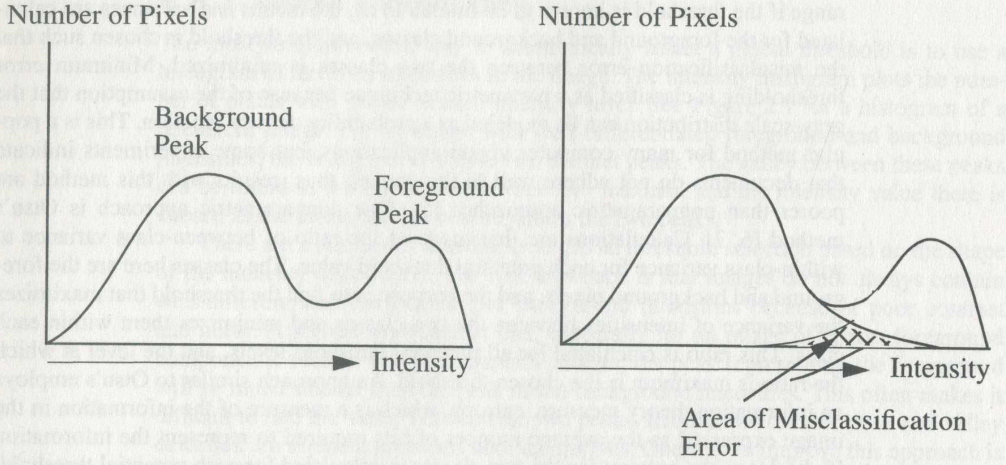


Figure 2.2 Illustration of misclassification error in thresholding. *Left*, intensity histogram showing foreground and background peaks; *right*, the tails of the foreground and background populations have been extended to show the intensity overlap of the two populations. This overlap makes it impossible to correctly classify all pixels using a single threshold. The minimum-error method of threshold selection minimizes the total misclassification error.

local pixel values taken for all pixels in the image (Figure 2.3). Mitchell and Gillies [13] describe a similar thresholding method where background white-level normalization is first done by estimating the white level and subtracting this level from the raw image. Segmentation of characters is accomplished by applying a range of thresholds and selecting the resulting image with the least noise content. Noise content is measured as the sum of areas occupied by components that are smaller and thinner than empirically determined parameters. From the results of binarization for different thresholds shown in Figure 2.1, one can see that the best threshold selection yields the least visible noise. The main problem with any adaptive binarization technique is the choice of window size. The window size should be large enough to guarantee that enough background pixels are included to obtain a good estimate of average value, but not so large as to average over non-uniform background intensities. Often, however, the features in the image vary in size, causing problems with fixed window size. To remedy this, domain dependent information can be used to ensure that the results of binarization give the expected features (a large blob of an ON-valued region is not expected in a page of smaller symbols, for instance). If the result is unexpected, then the window size can be modified and binarization applied again.

2.2.3 Choosing a Thresholding Method

Whether global or adaptive thresholding methods are used for binarization, one can never expect perfect results. Depending on the quality of the original, there may be

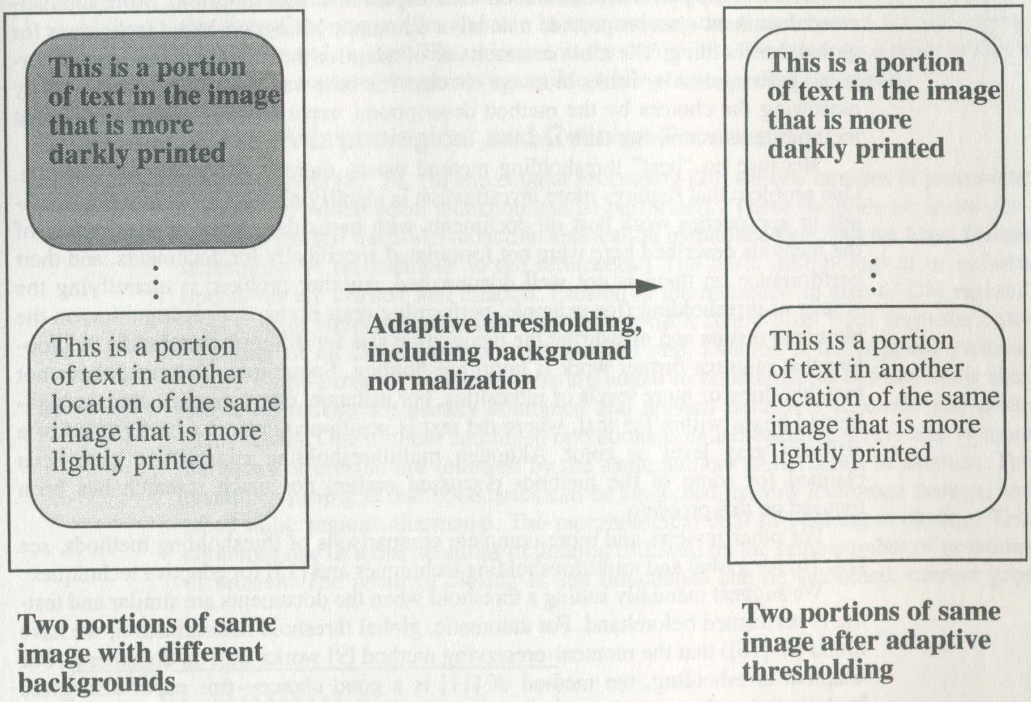

This is a portion of text in the image that is more darkly printed
This is a portion of text in another location of the same image that is more lightly printed
Adaptive thresholding, including background normalization
This is a portion of text in the image that is more darkly printed
This is a portion of text in another location of the same image that is more lightly printed
Two portions of same image with different backgrounds
Two portions of same image after adaptive thresholding

Figure 2.3 Diagram illustrating adaptive thresholding by background normalization. *Left*, original image has portions of text with different average background values; *right*, the image shows that the backgrounds have been eliminated leaving only ON and OFF values.

gaps in lines, ragged edges on region boundaries, and extraneous pixel regions of ON and OFF values. This is generally true with other document and image processing methods. The recommended procedure is to process as well as possible at each step of processing and to defer decisions that do not have to be made until later steps to avoid making irreparable errors. In later steps more information is available as a result of processing to that point, and this provides greater context and higher level descriptions to aid in making correct decisions and ultimately recognition. Deferment, when possible, is a principle appropriate for all stages of document analysis.

A number of different thresholding methods have been presented in this section—no single method is best for all image types and applications. For simpler problems where the image characteristics do not vary much within the image or across different images, the simpler methods will suffice. For more difficult problems of noise or varying image characteristics, more complex (and time-consuming) methods will usually be required. Commercial products vary in their thresholding capabilities. Today's scanners usually perform binarization with respect to a fixed threshold. More sophisticated document systems provide manual or automatic histogram-based techniques for global thresholding. The most common use of adaptive thresholding is in special-purpose systems used by banks to image checks. The best way to choose a method is by narrowing the choices by the method descriptions, experimenting with the different methods, and examining their results.

Because no "best" thresholding method exists, there is still room for research. One problem that requires more investigation is identifying which thresholding methods or approaches work best on documents with particular characteristics. Many of the methods described here were not formulated specifically for documents, and their performance on them is not well documented. Another problem is quantifying the results of thresholding (for example, performing optical character recognition on the binarized results and measuring the recognition rate for different thresholds). A problem that requires further work is multithresholding. Sometimes documents have not two, but three or more levels of intensities. For instance, many journals contain highlighted boxes within the text, where the text is positioned against a background of a different gray level or color. Although multithresholding capabilities have been claimed for some of the methods discussed earlier, not much research has been focused on this problem.

For other reviews and more complete comparisons of thresholding methods, see [14, 16] for global and multithresholding techniques and [15] for adaptive techniques.

We suggest manually setting a threshold when the documents are similar and testing is performed beforehand. For automatic, global threshold determination, we have found (in [16]) that the moment-preserving method [9] works well on documents. For adaptive thresholding, the method of [11] is a good choice—this paper also gives background and comparison on these adaptive methods. For multithresholding, the method in [7] is appropriate if the number of thresholds is known, and the method in [16] if not.

2.3 Noise Reduction

Keywords: filtering, noise reduction, salt-and-pepper noise, filling, morphological processing, cellular processing

After binarization, document images are usually filtered to reduce noise. Salt-and-pepper noise (also called impulse and speckle noise, or just dirt) is a prevalent artifact in poor quality document images (such as poorly thresholded faxes or poorly photocopied pages). This appears as isolated pixels or pixel regions of ON noise in OFF backgrounds or OFF noise (holes) within ON regions and as rough edges on characters and graphics components (Figure 2.4). The process of reducing noise is called filling. The most important reason to reduce noise is that extraneous features will otherwise cause subsequent errors in recognition. Furthermore, noise reduction reduces the size of the image file, and this in turn reduces the time required for subsequent processing and storage. The objective in the design of a filter to reduce noise is that it removes as much of the noise as possible while retaining all of the signal.

2.3.1 Morphological and Cellular Processing

Morphological [17, 18, 19] and cellular processing [20] are two families of processing methods by which noise reduction can be performed. (These methods are more general than for just the noise-reduction application mentioned here, but we leave further description of the methods to the references.) The basic morphological or cellular operations are erosion and dilation. Erosion is the reduction in size of ON regions. This is most simply accomplished by peeling off a single-pixel layer from the outer boundary of all ON regions on each erosion step. Dilation is the opposite process, where single-pixel, ON-valued layers are added to boundaries to increase their size. These operations are usually combined and applied iteratively to erode and dilate many layers. One of these combined operations is called opening, where one or more iterations of erosion are followed by the same number of iterations of dilation. The result of opening is that boundaries can be smoothed, narrow isthmuses broken, and small noise regions eliminated. The morphological dual of opening is closing. This combines one or more iterations of dilation followed by the same number of iterations of erosion. The result of closing is that boundaries can be smoothed, narrow gaps

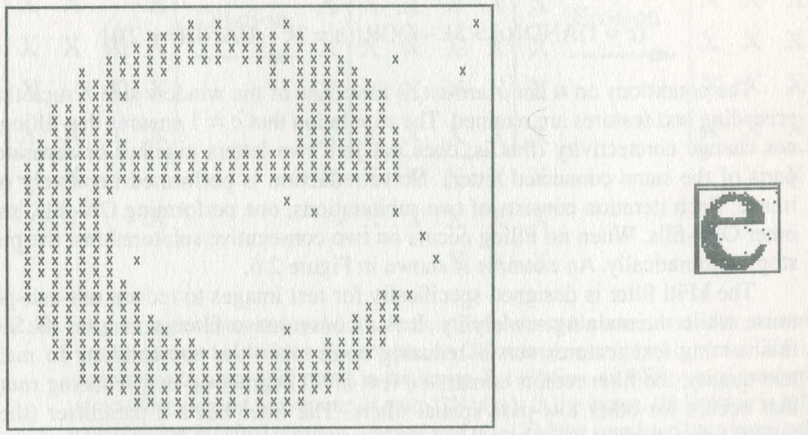


Figure 2.4 Illustration of letter *e* with salt-and-pepper noise. *Left*, the letter is shown with its ON and OFF pixels as Xs and blanks; *right*, the noisy character is shown.

joined, and small noise holes filled. See Figure 2.5 for an illustration of morphological operations.

2.3.2 Text and Graphics Noise Filters

For documents, more specific filters can be designed to take advantage of the known characteristics of the text and graphics components. In particular, we desire to maintain sharpness in these document components, not to round corners and shorten lengths, as some noise-reduction filters will do. Single-pixel islands, holes, and protrusions can be found by passing a 3 × 3 pixel window that matches these patterns over the image [21] and then filling it. For noise larger than one pixel, the kFill filter can be used [22].

We describe the kFill noise reduction method in more detail. Filling operations are performed within a k × k window that is applied in raster-scan order, centered on each image pixel. This window comprises an inside (k – 2) × (k – 2) region called the core, and the 4(k – 1) pixels on the window perimeter, called the neighborhood. The filling operation entails setting all values of the core to ON or OFF dependent upon pixel values in the neighborhood. The decision whether or not to fill with ON (OFF) requires that all core values must be OFF (ON) and depends on three variables determined from the neighborhood. For a fill value equal to ON (OFF), the n variable is the number of ON (OFF) pixels in the neighborhood, the c variable is the number of connected groups of ON pixels in the neighborhood, and the r variable is the number of corner pixels that are ON (OFF). Filling occurs when the following conditions are met:

$$(c = 1)\text{AND}\{(n > 3k - 4)\text{OR}[(n = 3k - 4)\text{AND}(r = 2)]\}$$

The conditions on n and r are set as functions of the window size k such that the preceding text features are retained. The stipulation that $c = 1$ ensures that filling does not change connectivity (that is, does not join two letters together or separate two parts of the same connected letter). Noise reduction is performed iteratively on the image. Each iteration consists of two subiterations, one performing ON-fills, and the other OFF-fills. When no filling occurs on two consecutive subiterations, the process stops automatically. An example is shown in Figure 2.6.

The kFill filter is designed specifically for text images to reduce salt-and-pepper noise while maintaining readability. It is a conservative filter, erring on the side of maintaining text features versus reducing noise when the two conflict. To maintain text quality, the filter retains corners on text of 90 degrees or less, reducing rounding that occurs for other low-pass spatial filters. The filter has a k parameter (the k in "kFill") that enables adjustment for different text sizes and image resolutions, thus enabling retention of small features such as periods and the stick ends of characters. Since this filter is designed for fabricated symbols, text, and graphics, it is not appropriate for binarized pictures where less regularly shaped regions and dotted shading

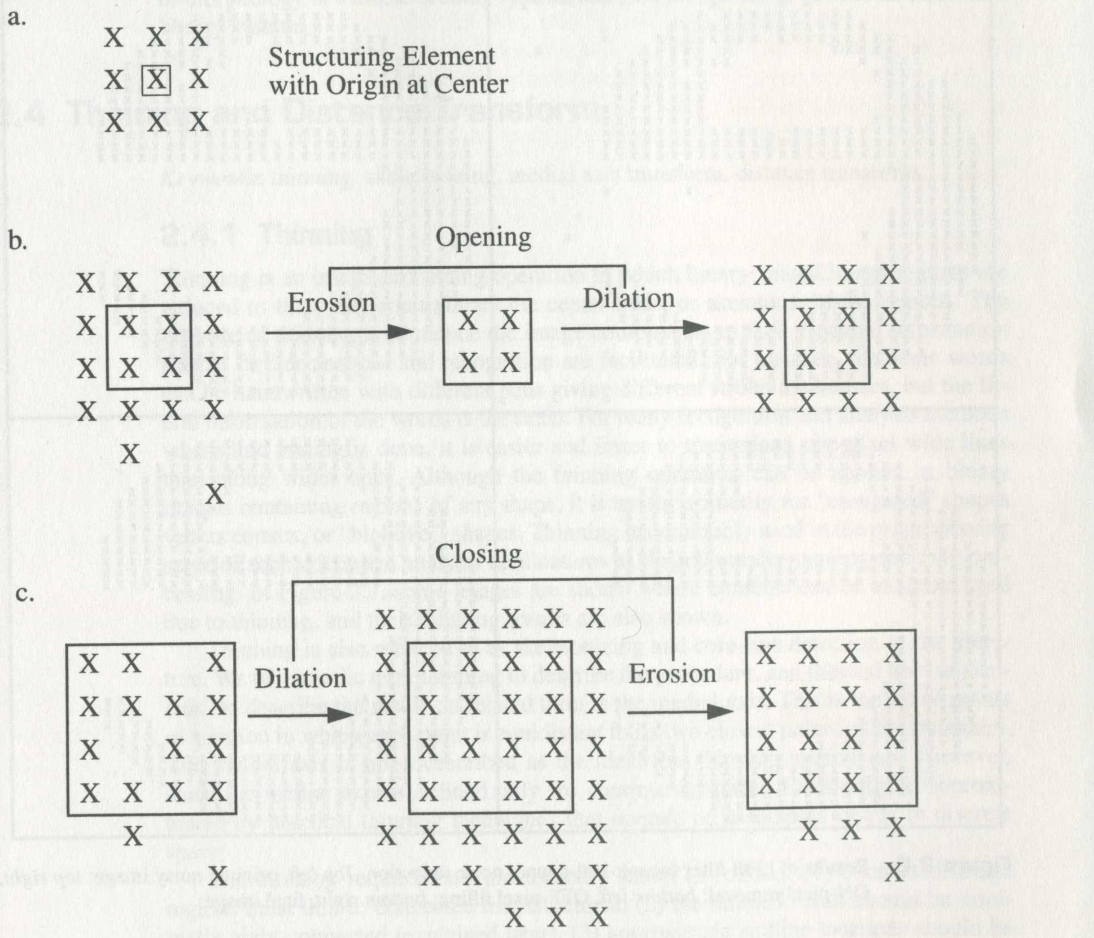


Figure 2.5 Morphological processing. (*a*) The structuring element is centered on each pixel in the image and pixel values are changed as follows. For erosion, an ON-valued center pixel is turned OFF if the structuring element is over one or more OFF pixels in the image. For dilation, an OFF-valued center pixel is turned ON if the structuring element is over one or more ON pixels in the image. (*b*) Erosion is followed by dilation; that combination is called opening. The isolated pixel and the spur have been removed in the final result. (*c*) Dilation is followed by erosion; that combination is called closing. The hole is filled, the concavity on the border is filled, and the isolated pixel is joined into one region in the final result.

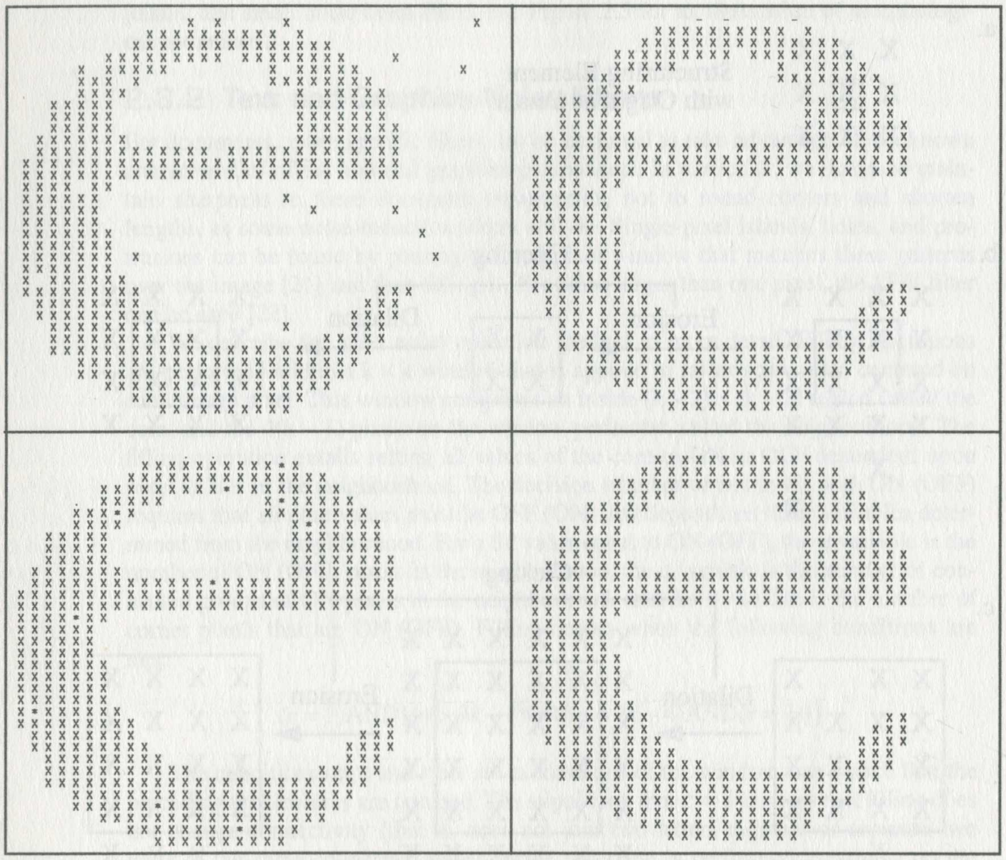


Figure 2.6 Results of kFill filter for salt-and-pepper noise reduction. *Top left*, original noisy image; *top right*, ON-pixel removal; *bottom left*, OFF-pixel filling; *bottom right*, final image.

(half-tone) are prevalent. A drawback of this filter — and of processes that iterate several times over the entire image —is that the processing time is expensive. Whether the cost of applying a filter such as this in the preprocessing step is justified depends on the input image quality and the tolerance for errors due to noise in subsequent steps.

Most document-processing systems perform rudimentary noise reduction by passing 3 × 3 filter masks across the image to locate isolated ON and OFF pixels. For more extensive descriptions of these techniques in document systems, see [23] for use of morphology in a music-reading system, and [24] for the use of kFill in an electronic library system.

2.4 Thinning and Distance Transform

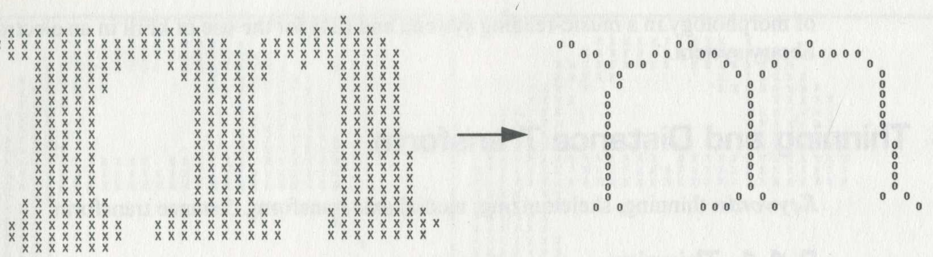
Keywords: thinning, skeletonizing, medial axis transform, distance transform

2.4.1 Thinning

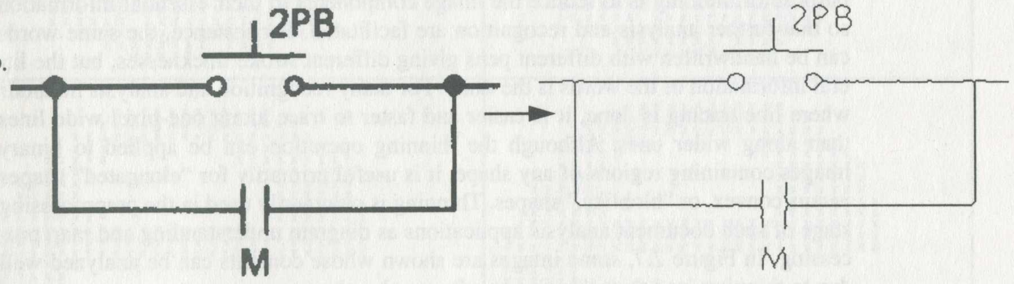
Thinning is an image-processing operation in which binary valued image regions are reduced to lines that approximate the center lines, or skeletons, of the regions. The purpose of thinning is to reduce the image components to their essential information so that further analysis and recognition are facilitated. For instance, the same words can be handwritten with different pens giving different stroke thicknesses, but the literal information of the words is the same. For many recognition and analysis methods where line tracing is done, it is easier and faster to trace along one-pixel wide lines than along wider ones. Although the thinning operation can be applied to binary images containing regions of any shape, it is useful primarily for "elongated" shapes versus convex, or "bloblike," shapes. Thinning is commonly used in the preprocessing stage of such document analysis applications as diagram understanding and map processing. In Figure 2.7, some images are shown whose contents can be analyzed well due to thinning, and their thinning results are also shown.

Thinning is also referred to as skeletonizing and core-line detection in the literature. We will use the term thinning to describe the procedure, and thinned line, or skeleton, to describe the results; a related term is the medial axis. This is the set of points of a region in which each point is equidistant to its two closest points on the boundary. The medial axis is often described as the ideal that thinning approaches. However, since the medial axis is defined only for continuous space, it can only be approximated by practical thinning techniques that operate on a sampled image in discrete space.

The thinning requirements are formally stated as follows: (1) connected image regions must thin to connected line structures, (2) the thinned result should be minimally eight-connected (explained later), (3) approximate endline locations should be maintained, (4) the thinning results should approximate the medial lines, and (5) extraneous spurs (short branches) caused by thinning should be minimized. That the results of thinning must maintain connectivity as specified by requirement 1 is essential. This guarantees an equal number of thinned connected line structures as the number of connected regions in the original image. By requirement 2, we stipulate that the resulting lines should always contain the minimal number of pixels that maintain eight-connectedness. (A pixel is considered eight-connected to another pixel if the second pixel is one of the eight closest neighbors to it.) Requirement 3 states that the locations of endlines should be maintained. Since thinning can be achieved by iteraa.



b.



c.

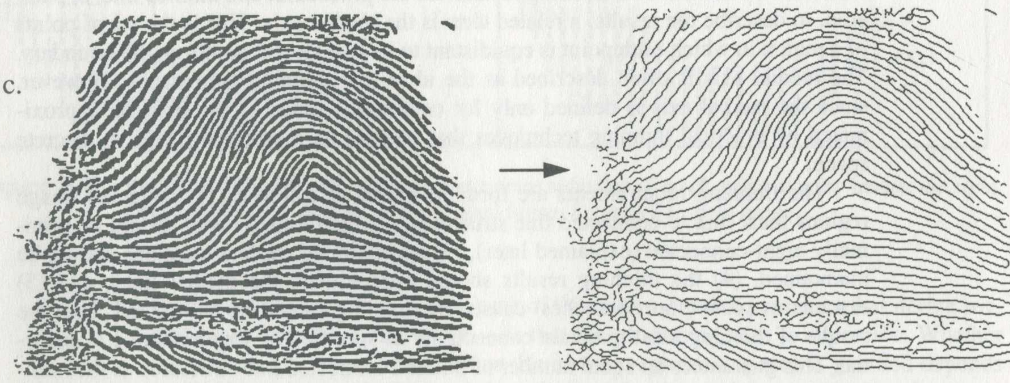


Figure 2.7 *Left*, original images; *right*, thinned image results. (*a*) The letter *m*, (*b*) a line diagram, (*c*) a fingerprint image.

tively removing the outer boundary pixels, it is important not to also iteratively remove the last pixels of a line. This would shorten the line and not preserve its location. Requirement 4 states that the resultant thin lines should best approximate the medial lines of the original regions. In digital space, the true medial lines can only be approximated. For instance, for a two-pixel wide vertical or horizontal, the true medial line should run at the half-pixel spacing along the middle of the original. Since it is impossible to represent this in digital image space, the result will be a single line running at one side of the original. With respect to requirement 5, noise should be minimized, but it is often difficult to say what is noise and what is not. We do not want spurs to result from every small bump on the original region, but we do want to recognize when a somewhat larger bump is a feature. Though some thinning algorithms have parameters to remove spurs, thinning and noise removal should be performed separately. Since one person's undesired spur may be another's desired short line, it is best to perform thinning first, then, in a separate process, remove any spurs whose length is less than a specified minimum.

The basic iterative thinning operation is to examine each pixel in an image within the context of its neighborhood region of at least 3×3 pixels and to "peel" the region boundaries, one pixel layer at a time, until the regions have been reduced to thin lines. (See [25] for basic 3×3 thinning, and [26] for generalization of the method to $k \times k$ sized masks.) This process is performed iteratively—on each iteration every image pixel is inspected, and single-pixel-wide boundaries that are not required to maintain connectivity or endlines are erased (set to OFF). In Figure 2.8, one can see how, on each iteration, the outside layer of one-valued regions is peeled off in this manner, and when no changes are made on an iteration, the image is thinned.

Unwanted in the thinned image are isolated lines and spurs off longer lines that are artifacts due to the thinning process or noise in the image. Some thinning methods, such as [27], require that the binary image is noise filtered before thinning because noise severely degrades the effectiveness and efficiency of this processing. Noise can never be totally removed, however, and, it is often difficult to distinguish noise from the signal in the earlier stages. An alternative approach is to thin after rudimentary noise reduction and then perform noise reduction with higher level information. Segments of image lines between endpoints and junctions are found and descriptive parameters (length, type as classified by junctions or endlines at the ends of the segment, average curvature, absolute location, and so on) are associated with them. This descriptive and contextual information is then used to remove the line artifacts.

Instead of iterating through the image for the number of times that is proportional to the maximum line thickness, thinning methods have been developed to yield the result in a fixed number of steps [27, 28, 29]. This is computationally advantageous when the image contains thick objects that would otherwise require many iterations. For these noniterative methods, skeletal points are estimated from distance measurements with respect to opposite boundary points of the regions (see Section 2.4.2 on distance transformation). Some of these methods require joining the line segments after thinning to restore connectivity and require a parameter estimating maximum

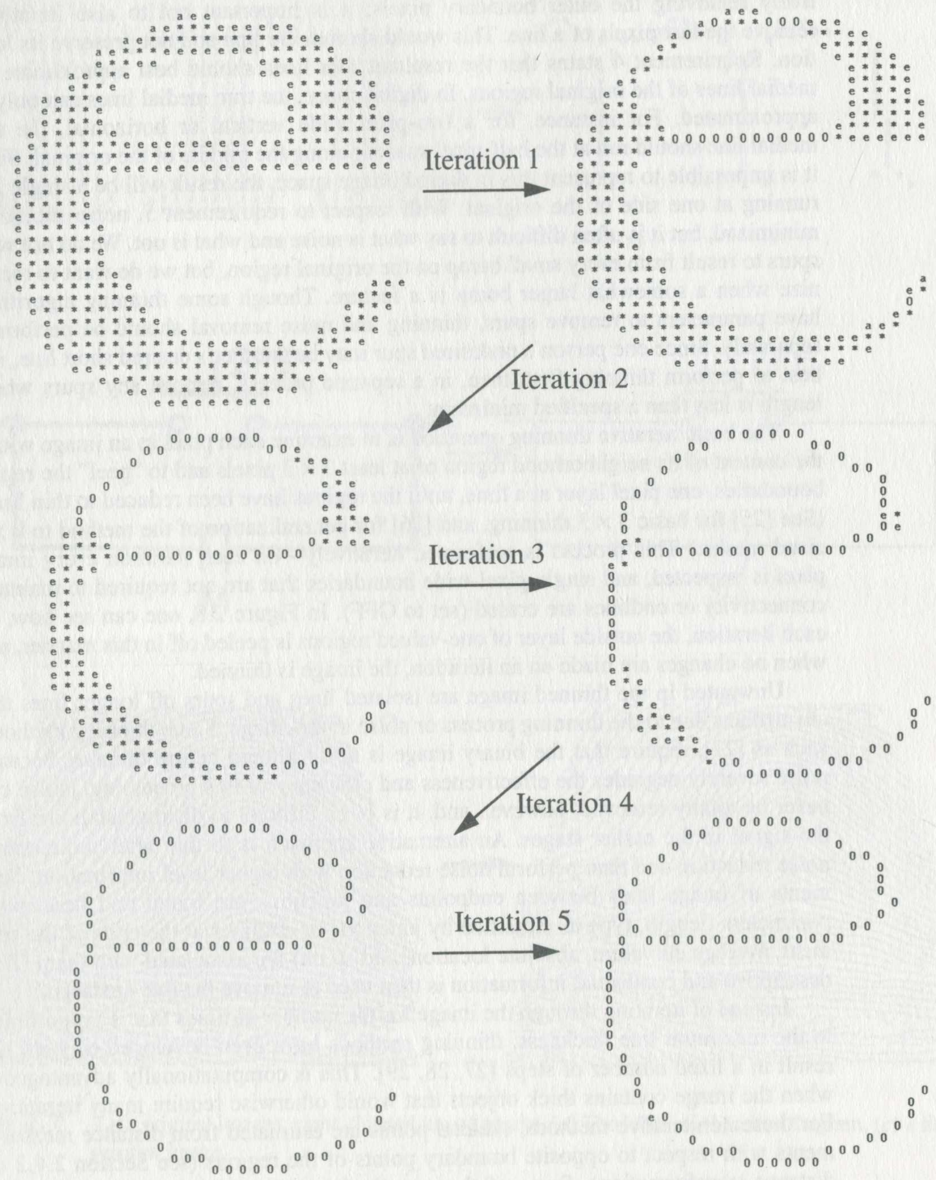


Figure 2.8 Sequence of five iterations of thinning on letter *e*. On each iteration a layer of the outer boundaries is peeled off. On iteration 5, the end is detected because no pixels change.

thickness of the original image lines so that the search for pairs of opposite boundary points is spatially limited. In general, compared to iterative methods, these noniterative thinning methods are less regularly repetitive, not limited to local operations, and less able to be pipelined; and these factors make their implementation in special-purpose hardware less appropriate.

Algorithms have been developed for extracting thin lines directly from gray-level images of line drawings by tracking along gray-level ridges—that is, without the need for binarization [30]. These have the advantage of being able to track along ridges whose peak intensities vary throughout the image such that binarization by global thresholding would not yield connected lines. A problem, however, with tracking lines on gray-scale images is following false ridges (the gray-scale equivalent of spurs), which results in losing track of the main ridge or requires computationally expensive backtracking. Binarization and thinning are the methods most commonly used for document analysis applications because they are well understood and relatively simple to implement.

For recent overview papers on thinning, see [31, 32]; for thinning applied specifically to documents, see [33].

2.4.2 Distance Transformation

The distance transform is a binary image operation in which each pixel is labeled by the shortest distance from it to the boundary of the region within which it is contained. One way this can be used is to determine the shortest path from a given interior point to the boundary. It can also be used to obtain the thinned image by retaining only ridges of maximum local distance measures. This thinned image, complete with distance values, has more information than simply the thinned image without distance information. It can be used as a concise and descriptive representation of the original image from which line widths can be obtained or the original image can be reconstructed. Results of the distance transform are shown in Figure 2.9.

Two approaches are used to obtain the distance transform. The first approach is similar to the iterative thinning method described earlier. On each iteration, boundaries are peeled from regions. Instead of setting each erased pixel value to OFF, they are set to the distance from the original boundary. Therefore, on the first iteration (examining 3×3 sized masks), erased boundaries are set to zero. On the second iteration, any erased core pixels will have a distance of 1 for vertical or horizontal distance to a boundary point or $\sqrt{2}$ for diagonal distance to a boundary point. On the third and subsequent iterations, each pixel's distance value is calculated as the sum of the smallest distance value of a neighboring pixel plus its distance to that neighbor. When a core can no longer be thinned, it is labeled with its distance to the closest boundary.

The second approach requires a fixed number of passes through the image. To obtain the integer approximation of the Euclidean distance, two passes are necessary. The first pass proceeds in raster order, from the top row to the bottom row, left to right on each row. The distances are propagated in a manner similar to the preceding approach, but because the direction of the raster scan is from top left to bottom right,

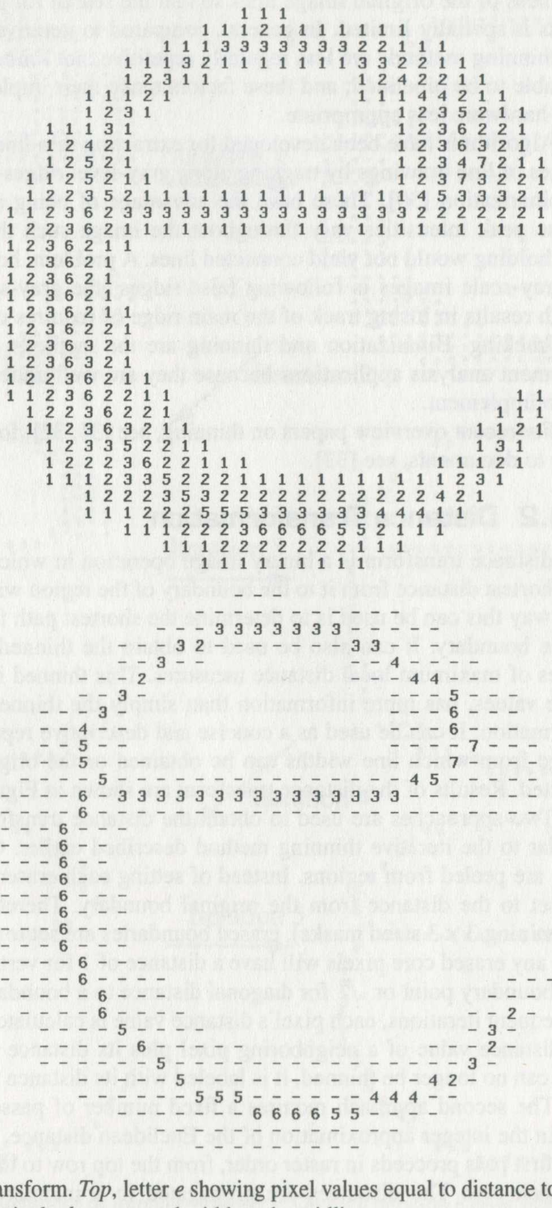


Figure 2.9 Distance transform. *Top*, letter *e* showing pixel values equal to distance to closest border, except for midline pixels; *bottom*, total width at the midline.

these first iteration values are only intermediate values—they only contain distance information from above and to the left. The second pass proceeds in reverse raster order, from bottom right to top left, where the final distance values are obtained taking into account the distance information from below and to the right as well. For further treatments of distance transformations, see [34, 35, 36].

The iterative method is used if iterative thinning is desired. As mentioned earlier, thinning is only appropriate for elongated regions, and if the distance transform of an image containing thick lines or more convex regions is desired, the fixed-pass method is more appropriate. The fixed-pass method can also be used as a first step toward thinning. For all of these distance transformation methods, the distance must not exceed the pixel word size, usually a byte, since distance is stored in the pixel; furthermore, floating point distance is approximated by integer numbers, usually by scaling the floating point number up (for example, 1.414 would become 14). Word size consideration is usually not a problem for images of relatively thin, elongated regions but may be a problem for larger regions.

Thinning is available on most commercial graphics analysis systems. The particular method varies—many are available—but the method is usually iterative and uses 3×3 or 3×4 sized masks. Most systems take advantage of a fast table lookup approach for the mask operations. These are implemented in software or in hardware for more specialized (faster and more expensive) machines.

2.5 Chain Coding and Vectorization

Keywords: chain code, Freeman chain code, primitives chain code (PCC), line and contour compression, topological feature detection, vectorization

2.5.1 Chain Coding

When objects are described by their skeletons or contours, they can be represented more efficiently than simply by ON and OFF valued pixels in a raster image. One common way to do this is by chain coding, where the ON pixels are represented as sequences of connected neighbors along lines and curves. Instead of storing the absolute location of each ON pixel, the direction from its previously coded neighbor is stored. A neighbor is any of the adjacent pixels in the 3×3 pixel neighborhood around that center pixel (Figure 2.10). Coding by direction has two advantages over absolute coordinate location. One is in storage efficiency. For commonly sized images larger than 256×256, the coordinates of an ON-valued pixel are usually represented as two 16-bit words; in contrast, for chain coding with eight possible directions from a pixel, each ON-valued pixel can be stored in a byte or even packed into 3 bits. A more important advantage in this context is that, since the chain coding contains information on connectedness within its code, this can facilitate further processing, such as

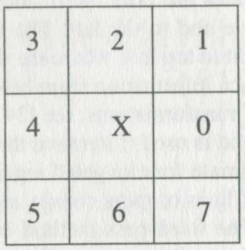


Figure 2.10 3 × 3 pixel region with center pixel denoted as X, showing codes for chain directions from center pixel to each of eight neighbors: 0 (east), 1 (northeast), 2 (north), 3 (northwest), and so on.

smoothing of continuous curves and analysis, such as feature detection of straight lines.

The definition of connected neighbors that we will use here is called eight-connected; that is, a chain can connect from one pixel to any of its eight closest neighbors in directions 0 to 7 in Figure 2.10. Other definitions of connectedness are also used in the literature. The most common is four-connected, where a pixel can be connected to any of its four closest neighbors in the directions 0, 2, 4, or 6. An advantage of the four-connected chain is that each of its four chain directions has the same distance of one pixel spacing (eight-connected chains have two distances: 1 and $\sqrt{2}$ pixel spacings). The primary advantage of the eight-connected chain is that it more closely represents diagonal lines and portions of lines; that is, without the horizontal and vertical stair-steps to which the four-connected code is limited. Eight-connected codings also yield more concise representations.

The Freeman chain code is a widely used chain coding method [37]. Coding is accomplished in the following manner. A raster search is made from the top left of the image to the bottom right, examining each pixel. When an ON-valued pixel is found, the coordinate location of this pixel is stored, that pixel is set to OFF in the image, and chain coding is begun. The direction code is stored for each connection from the current pixel to a neighboring pixel, the current pixel is set to OFF, and this is done for all connected pixels until the end of a line or until a closed loop rejoins. If there is a branch in the line, one of the branching neighbors is chosen arbitrarily. The end of the chain is indicated by adding a code that is the inverse direction of the previous code (for example, 0, 4; or 1, 5; and so forth), and since this is otherwise impossible, this indicates the chain end. See Figure 2.11 for an example of a Freeman chain coded line structure.

Though the Freeman chain code is highly effective for compression of line images, it was designed for contours, without any provision for maintaining branching line structures (each branch is coded as a separate chain). This is fine for compression, but for image analysis it is important to retain the complete line structure with all

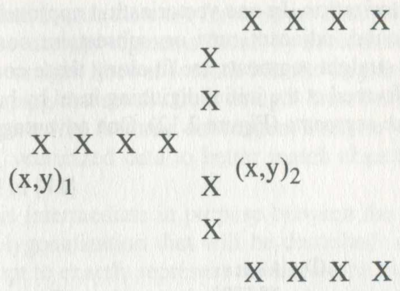


Figure 2.11 Pixels of a branching line structure.The Freeman chain code for this is $\{(x,y)_1$, 0, 0, 0, 1, 2, 1, 0, 0, 0, 4, $(x,y)_2$, 6, 7, 0, 0, 0, 4$\}$, where the top branch is coded first, then the bottom beginning with the pixel at $(x,y)_2$.

its branches and to know the topology at each junction. In [38], the skeleton is coded with pixel values of 0, 1, 2, and 3 for background pixels, terminal (end-of-line) pixels, intermediate pixels, and junction pixels respectively. This, combined with chaining, allows line segments and their interconnections to be easily determined.

Another method, the primitives chain code (PCC) also preserves topological features [39]. PCC contains codes for the following features: ends of lines, bifurcation and cross junctions, and breaks indicating the beginning of coding within a closed contour. With these additional features, subsequent pattern recognition steps are facilitated. PCC usually results in higher compression than the other chain code techniques because it efficiently limits the number of code words to the number of eight-connected possibilities and packs them efficiently. The PCC has been applied to analysis of fingerprints, maps, and engineering diagrams, all of which contain branches as important features. PCC coding is accomplished in a manner similar to Freeman chain coding; that is, a raster scan for ON-valued pixels is first done, and any lines are followed and encoded. The difference is that features are also encoded, and because of the presence of codes for these features, different line topologies can be recognized from the code. For the example in Figure 2.11, the PCC code is: $(x,y)_1$, 41, **b**, 15, 41, **e**, 61, 41, **e**, where $(x,y)_1$ is the starting point, followed by the chain and feature codes. The bold characters indicate PCC features and the non-bold numbers indicate portions of up to three chain directions between the features. Here, the features are seen easily as a branch junction (*b*) followed by two endlines (*e*). The other codes are table lookup values for the connections between features: 41 is equivalent to {0,0,0} in Freeman chain coding, 15 is equivalent to {1,2,1}, and 61 is equivalent to {6,7,0}.

2.5.2 Vectorization

An alternative to thinning and chain coding is to represent image lines by the straight line segments that can be drawn within the original thicker lines. This approach is called vectorization. In one vectorization approach, [40], horizontal runs within lines are first found, adjacent runs on subsequent scan lines are grouped together, and piecewise-straight segments are fit along these connected regions. In [41], vectorization is performed at the initial digitizing level by hardware that locates and tracks long straight line segments (Figure 2.12). One advantage of vectorization is that, since long

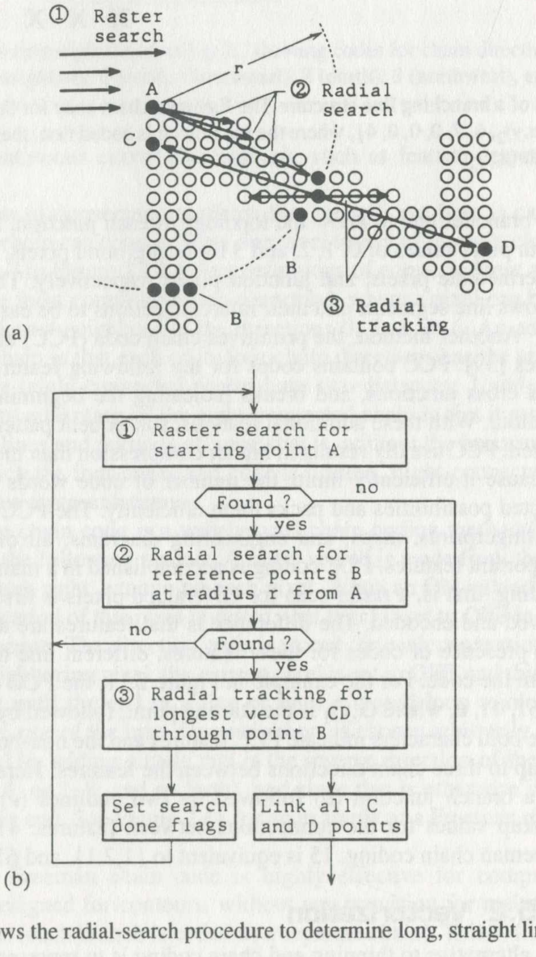

① Raster search
r
A
② Radial search
C
D
B
B
③ Radial tracking
(a)
① Raster search for starting point A
Found ?
no
yes
② Radial search for reference points B at radius r from A
no
Found ?
yes
③ Radial tracking for longest vector CD through point B
Set search done flags
Link all C and D points
(b)

Figure 2.12 The diagram shows the radial-search procedure to determine long, straight lines for vectorization [41].

lines are searched and found, there are fewer spurs than usually result by thinning and chain coding. These straight segments found by vectorization will not usually correspond to complete straight lines as intended in the original drawing. This is partially due to the stair-stepping that results from straight lines being digitized at slight angles. To determine complete line features, subsequent analysis is necessary. An example of post-processing of vectorized data to better match objects found against the object model is described in [42]

Vectorization is intermediate in purpose between the thinning and chain coding procedures and polygonalization that will be described in Chapter 3. Thinning and chain coding attempt to exactly represent the line paths, polygonalization attempts to approximate line features, and vectorization yields something between the two. If the vectorization tolerance parameter is zero, the results are closest to those of thinning and chain coding. If the tolerance is such that a piecewise straight approximation of straight and curved lines results, this is similar to polygonalization. In practice, the term *vectorization* is often used loosely for techniques that span this range. To avoid this confusion we use *chain coding* and *polygonalization* to describe families of methods and *vectorization* only for the so-called hardware vectorizers mentioned here.

A final note should be made on automatic and human-assisted vectorization. Often the image quality is such that automatic line extraction is too error-prone to be useful. For instance, for conversion of engineering drawings to CAD format from older blueprints, the cost of a human operator correcting all the errors made by the computer may be higher than manually entering the entire diagram in the first place. For cases such as this, a computer can be used to facilitate manual conversion. One example of this is Fastrak [43], an interactive line-following digitizer in which a human operator simply "leads" the computer along the path of the line using an input device such as a mouse. Most practical commercial systems for diagram entry employ a combination of image analysis and computer-aided manual correction.

Chain coding or vectorization is performed in most commercially available systems for storing and analyzing graphics documents. The Freeman code is the most common, though systems often incorporate proprietary vectorization techniques. Refer to [42] for vectorization for map processing.

2.6 Binary Region Detection

Keywords: binary segmentation, region coloring, blob detection, connected component labeling, contour detection, line adjacency graph (LAG)

Before feature-level analysis can take place, segmentation must be performed to detect individual regions (also called objects or blobs) in the image. For gray-scale and color images, detection is sometimes difficult because the objects may blend into the background or there may be overlap. For binary images, however, it is a straightforward procedure to find each group of neighboring ON pixels. Thus, the character, *a* comprises one region, and the character *i* two regions. A dashed line is composed of as many regions. Once regions have been found, features can be determined from them, and recognition can be made of the region as one or part of a particular character or graphics component.

Region detection is analogous to chain coding preceded by thinning since the objective of both procedures is to yield a single representation of a group of neighboring ON pixels. The difference between the two procedures is their applicability. As has been mentioned, thinning and chain coding can be applied to any shape but are most useful for elongated shapes. Region detection can also be applied to any shape, but it is especially useful for rounder shapes where the results from thinning are less useful; for example, a circular disk will thin to a single point (under perfect thinning conditions), and the size information will be lost in this thinned image; whereas region detection will retain this size information. Region detection is also useful for hole detection. For instance, thinning an image of Swiss cheese will result in many connected lines representing the cheese connections; however, this may not be as useful as the result of contour detection that will have one contour for the boundary of the cheese and contours for each of the holes. As a counterexample, for the letter *H*, thinning will yield the pertinent topological information that the symbol is made up of two vertical lines joined by a horizontal line, whereas region detection will yield only a single region with no topological information and will require additional feature detection for character recognition. Since many document components are made from line strokes, thinning and chain coding are often used; however, if the purpose is just to locate document components, or if the objects are truly blob-shaped, region detection is appropriate.

2.6.1 Contour Detection

Contour detection can be thought of as a reciprocal operation of thinning. Whereas thinning yields the inside skeletons, contour detection yields the outside boundaries, or contours. Since a single contour envelopes a single region, contour detection can be used for region detection. Contours are composed of boundary, ON-valued pixels that border OFF-valued pixels. These contours can be found easily by examining each pixel within a 3×3 window, and if the center pixel is ON, and at least one of its neighborhood pixels is OFF, the center pixel is a contour pixel and it is set to ON; all other pixels are set to OFF. Each resulting contour can then be chain coded to represent each region. Usually chain coding is done in one direction around ON regions (counterclockwise) and in the opposite direction (clockwise) around holes. The result of contour detection is illustrated in Figure 2.13. (See [44, 45] for contour tracing algorithms.)

The contour image can be used in a number of ways. For instance, the number of contours gives the number of regions (both ON-valued regions and OFF-valued holes within ON-valued regions). The centroid of the boundary pixels gives a measure of

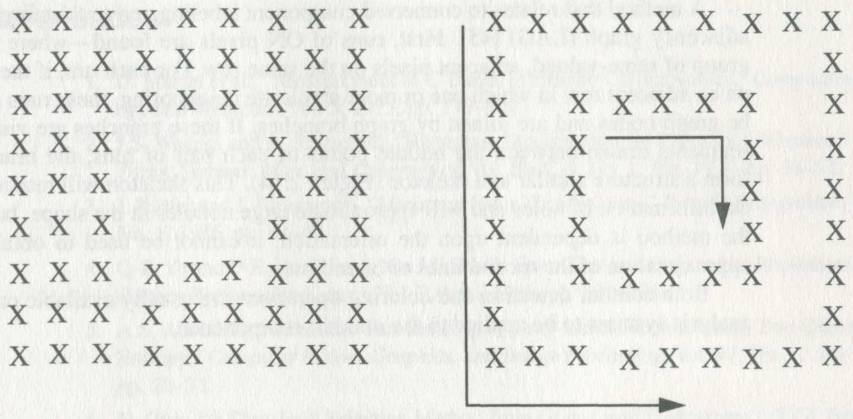


Figure 2.13 *Left*, region, with ON-valued pixels represented by Xs; *right,* contour of this region, where only the outer and inner boundaries remain ON. The arrows around the contour show the direction of coding.

the region location. The length of a contour indicates the enclosed region size. The length and enclosed area can be used to give a measure of how elongated or "fat" the region is. Curvature and corner features can be determined from the contour to determine the region shape. (For other feature detection methods applicable to contours, see Chapter 3.)

2.6.2 Region Labeling

Regions can also be found by other techniques based on interior (versus contour) pixels. One way to locate regions is to perform connected component labeling, or region coloring, which results in each region of an image being assigned a different label or color [46, 47]. The method involves a two-pass process where pixels are individually labeled. The image is first scanned in raster order, and each pixel is examined. For each ON-valued pixel, the previously labeled pixels to the left and above are examined. If none is ON, then the pixel is set to a new label value. If one of these is ON, then the current pixel is given the same label as that pixel. If more than one are ON, then the pixel is set to one of the labels, and all labels are put in an equivalence class (that is, they are stored for later merging). At the end of this pass, the number of connected components is the number of equivalence classes, plus the number of labels not in an equivalence class. The second pass consists first of merging all labels in each equivalence class to be the same label and then reassigning all labeled pixels in the image to these final labels.

A method that relates to connected component labeling and to thinning is the line adjacency graph (LAG) [45]. First, runs of ON pixels are found—where a run is a group of same-valued, adjacent pixels on the same row. For each run, if there is a run on an adjacent row in which one or more pixels are neighboring, these runs are said to be graph nodes and are joined by graph branches. If these branches are visualized as segments drawn between the middle points of each pair of runs, the branches will form a structure similar to a skeleton (Figure 2.14). This skeleton will retain topological information of holes and will approximate large nodules in the shape, but because the method is dependent upon the orientation, it cannot be used to obtain a good approximation of the medial lines or of endlines.

Both contour detection and coloring operations are usually available on graphics analysis systems to be applied to the nonthin components.

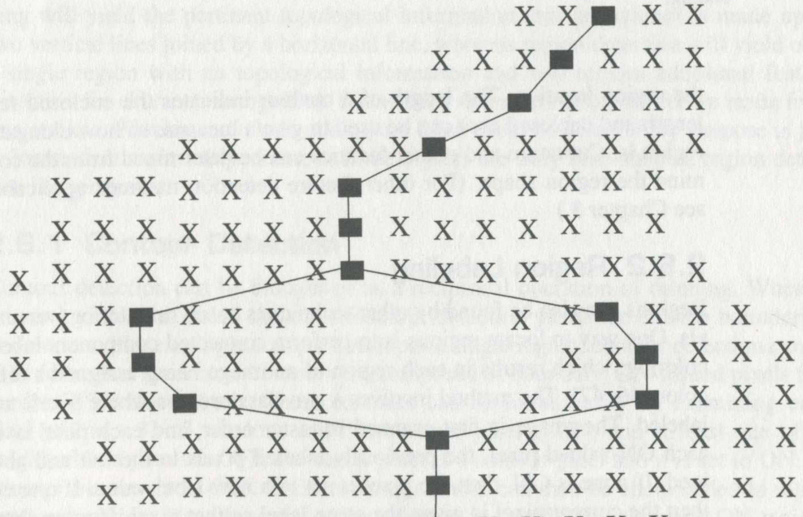


Figure 2.14 Line adjacency graph of region. The ON pixels are shown as Xs. The middle of each row of ON pixels is shown as a filled box. The lines that join these midpoints of each ON row shows the line adjacency graph.

2.7 References

1. D. Mason et al., "Measurement of C-Bands in Human Chromosomes," *Computers in Biology and Medicine*, Vol. 5, 1975, pp. 179–201.
2. J.S. Weszka and A. Rosenfeld, "Histogram Modification for Threshold Selection," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. SMC-9, No. 1, Jan. 1979, pp. 38–52.
3. J. Kittler and J. Illingworth, "Minimum Error Thresholding," *Pattern Recognition*, Vol. 19, No. 1, 1986, pp. 41–47.
4. Q-Z. Ye and P-E. Danielson, "On Minimum Error Thresholding and its Implementations," *Pattern Recognition Letters*, Vol. 7, April 1988, pp. 201–206.
5. A.S. Abutaleb, "Automatic Thresholding of Gray-Level Pictures Using Two-Dimensional Entropy," *Computer Vision, Graphics, and Image Processing*, Vol. 47, No. 1, July 1989, pp. 22–32.
6. N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. SMC-9, No. 1, Jan. 1979, pp. 62–66.
7. S.S. Reddi, S.F. Rudin, and H.R. Keshavan, "An Optimal Multiple Threshold Scheme for Image Segmentation," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. SMC-14, No. 4, July–Aug, 1984, pp. 661–665.
8. J.N. Kapur, P.K. Sahoo, and A.K.C. Wong, "A New Method for Gray-Level Picture Thresholding Using the Entropy of the Histogram," *Computer Vision, Graphics, and Image Processing*, Vol. 29, No. 29, Mar. 1985, pp. 273–285.
9. W-H. Tsai, "Moment-Preserving Thresholding: A New Approach," *Computer Vision, Graphics, and Image Processing*, Vol. 29, No. 29, Mar. 1985, pp. 377–393.
10. R.G. Casey and K.Y. Wong, "Document Analysis Systems and Techniques," in *Image Analysis Applications*, R. Kasturi and M.M. Trivedi, eds., Marcel Dekker, New York, 1990, pp. 1–36.
11. M. Kamel and A. Zhao, "Extraction of Binary Character/Graphics Images from Grayscale Document Images," *CVGIP: Graphical Models and Image Processing*, Vol. 55, No. 3, 1993, pp. 203–217.
12. J.M. White and G.D. Rohrer, "Image Thresholding for Optical Character Recognition and Other Applications Requiring Character Image Extraction," *IBM J. Res. Development*, Vol. 27, No. 4, July 1983, pp. 400–411.
13. B.T. Mitchell and A.M. Gillies, "A Model-Based Computer Vision System for Recognizing Handwritten ZIP Codes," *Machine Vision and Applications*, Vol. 2, No. 4, 1989, pp. 231–243.
14. P.K. Sahoo et al., "A Survey of Thresholding Techniques," *Computer Vision, Graphics, and Image Processing*, Vol. 41, No. 2, Feb. 1988, pp. 233–260.
15. O.D. Trier and T. Taxt, "Evaluation of Binarization Methods for Document Images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 17, No. 3, Mar. 1995, pp. 312–315.

16. L. O'Gorman, "Binarization and Multi-Thresholding of Document Images Using Connectivity," *CVGIP: Graphical Models and Image Processing*, Vol. 56, No. 6, Nov. 1994, pp. 494–506.

17. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.

18. R.M. Haralick, S.R. Sternberg and X. Zhuang, "Image Analysis Using Mathematical Morphology," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 9, July 1987, pp. 532–550.

19. R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision,* Addison-Wesley, Reading, Mass., 1992.

20. K. Preston, Jr. et al., "Basics of Cellular Logic with Some Applications in Medical Image Processing," *Proc. IEEE*, May 1979, pp. 826–855.

21. C-C. Shih and R. Kasturi, "Generation of a Line-Description File for Graphics Recognition," *Proc. SPIE Conf. on Applications of Artificial Intelligence*, Vol. 937, 1988, pp. 568–575.

22. L. O'Gorman, "Image and Document Processing Techniques for the RightPages Electronic Library System," *Proc. Int'l Conf. Pattern Recognition (ICPR)*, IEEE CS Press, Los Alamitos, Calif., 1992, pp. 260–263.

23. B.R. Modayur et al., "MUSER—A Prototype Music Score Recognition System Using Mathematical Morphology," *Machine Vision and Applications*, Vol. 6, No. 2, 1993, pp. 2–3.

24. G. Story et al., "The RightPages Image-Based Electronic Library for Alerting and Browsing," *Computer*, Vol. 25, No. 9, Sept. 1992, pp. 17–26.

25. C.J. Hilditch, "Linear Skeletons from Square Cupboards," *Machine Intelligence,* Vol. 4, 1969, pp. 403–420.

26. L. O'Gorman, "k × k Thinning," *Computer Vision, Graphics, and Image Processing*, Vol. 51, 1990, pp. 195–215.

27. C. Arcelli and G. Sanniti di Baja, "A Width-Independent Fast Thinning Algorithm," *IEEE Trans. Pattern Recognition and Machine Intelligence*, Vol. PAMI-7, No. 4, July 1985, pp. 463–474.

28. C. Arcelli, and G. Sanniti di Baja, "A One-Pass Two-Operation Process to Detect the Skeletal Pixels on the 4-Distance Transform," *IEEE Trans. Pattern Recognition and Machine Intelligence*, Vol. 11, No. 4, July 1989, pp. 411–414.

29. R.M.K. Sinha, "A Width-Independent Algorithm for Character Skeleton Estimation," *Computer Vision, Graphics, and Image Processing*, Vol. 40, No. 3, Dec. 1987, pp. 388–397.

30. L.T. Watson et al., "Extraction of Lines and Drawings from Grey Tone Line Drawing Images," *Pattern Recognition*, Vol. 17, No. 5, 1984, pp. 493–507.

31. L. Lam, S-W. Lee, and C.Y. Suen, "Thinning Methodologies—A Comprehensive Survey," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 14, No. 9, Sept. 1992, pp. 869–885.

32. L. Lam and C.Y. Suen, "An Evaluation of Parallel Thinning Algorithms for Character Recognition," *IEEE Trans. Pattern Recognition and Machine Intelligence*, Vol. 17, No. 9, Sept. 1995, pp. 914–919.
33. U. Eckhardt et al., "Thinning for Document Processing," *Proc. Int'l Conf. Document Analysis and Recognition*, 1991, pp. 490–498.
34. G. Borgefors, "Distance Transformations in Arbitrary Dimensions," *Computer Vision, Graphics, and Image Processing*, Vol. 27, No. 3, Sept. 1984, pp 321–345.
35. G. Borgefors, "Distance Transformations in Digital Images," *Computer Vision, Graphics, and Image Processing*, Vol. 34, No. 3, June 1986, pp. 344–371.
36. C. Arcelli et al., "Ridge Points in Euclidian Distance Maps," *Pattern Recognition Letters*, Vol. 13, No. 4, Apr. 1992, pp. 237–243.
37. H. Freeman, "Computer Processing of Line Drawing Images," *Computing Surveys*, Vol. 6, No. 1, 1974, pp. 57–98.
38. J.F. Harris et al., "A Modular System for Interpreting Binary Pixel Representations of Line-Structured Data," in *Pattern Recognition: Theory and Applications*, J. Kittler, K.S. Fu, L.F. Pau (eds.), D. Reidel, Boston, 1982, pp. 311–351.
39. L. O'Gorman, "Primitives Chain Code," in *Progress in Computer Vision and Image Processing*, A. Rosenfeld and L. G. Shapiro, eds., Academic Press, San Diego, 1992, pp. 167–183.
40. K. Ramachandran, "A Coding Method for Vector Representation of Engineering Drawings," *Proc. IEEE*, Vol. 68, 1980, pp. 813–817.
41. M. Ejiri et al., "Automatic Recognition of Engineering Drawings and Maps," in *Image Analysis Applications*, R. Kasturi and M.M. Trivedi, eds., Marcel Dekker, New York, 1990.
42. O. Hori and A. Okazaki, "High Quality Vectorization Based on a Generic Object Model," in *Structured Document Image Analysis*, H.S. Baird, H. Bunke, and K. Yamamoto, eds., Springer-Verlag, Berlin, 1992, pp. 325–339.
43. N.C. Fulford, "The Fastrak Automatic Digitizing System," *Pattern Recognition*, Vol. 14, No. 1, Jan. 1981, pp. 65–74.
44. R. Cederberg, "On the Coding, Processing, and Display of Binary Images," PhD dissertation No. 57, Linkoping Univ., Dept. Electrical Eng., Linkoping, Sweden, 1980.
45. T. Pavlidis, *Algorithms for Graphics and Image Processing*, Computer Science Press, Rockville, Md, 1982, pp. 116–120 (LAG), 142–148 (contour detection).
46. C. Ronse and P.A. Divijver, *Connected Components in Binary Images: The Detection Problem*, Research Studies Press Ltd., Letchworth, England, 1984.
47. M.B. Dillencourt, H. Samet, and M. Tamminen, "A General Approach to Connected-Component Labeling for Arbitrary Image Representations," *J. Association for Computing Machinery*, Vol. 39, No. 2, Apr. 1992, pp. 253–280.

more compact than storing the document as a file of pixels. For example, just as an English letter is stored as its 8-bit ASCII representation in lieu of its larger-size image, for a compression of two to three orders of magnitude, a graphics symbol, such as a company logo or electrical "AND" gate, can also have a similarly compact "code word" that essentially indexes the larger-sized image. Document image analysis can be important when the original document is produced by computer as well. Anyone who has dealt with transport and conversion of computer files knows that compatibility can rarely be taken for granted. Because of the many different languages, proprietary systems, and changing versions of CAD and text-formatting packages that are used, incompatibility is especially true in this area. Because the formatted document—that viewed by humans—is semantically the same independent of the language of production, this form is a "protocol-less protocol." If a document system can translate between different machine-drawn formats, the next objective is to translate from hand-drawn graphics. This is analogous to handwriting recognition and text recognition in OCR. When machines can analyze complex hand-drawn diagrams accurately and quickly, the graphics recognition problem will be solved, but there is still much opportunity for research before this goal will be reached.

A common sequence of steps taken for document image analysis of graphics interpretation is similar to that for text. Preprocessing, segmentation, and feature extraction methods such as those described in earlier chapters are first applied. An initial segmentation step that is generally applied to a mixed text-graphics image is that of text and graphics separation. An algorithm specifically designed for separating text components in graphics regions irrespective of their orientation is described in [1]. This is a Hough transform-based technique that uses the heuristic that text components are colinear. Once text is segmented, typical features extracted from a graphics image include straight lines, curves, and filled regions. After feature extraction, pattern recognition techniques are applied, both structural pattern recognition methods to determine the similarity of an extracted feature to a known feature using geometric and statistical means, and syntactic pattern recognition techniques to accomplish this same task using rules (a grammar) on context and sequence of features. After this mid-level processing, these features are assembled into entities with some meaning—or semantics—that is dependent on the domain of the particular application. Techniques used for this include pattern matching, hypothesis and verification, and knowledge-based methods. The semantic interpretation of a graphics element may be different depending on domain; for instance, a line may be a road on a map or an electrical connection of a circuit diagram. Methods at this so-called high level of processing are sometimes described as artificial intelligence techniques.

Most commercial OCR systems will recognize long border and table lines as being different from characters so will not attempt to recognize them as characters. Graphics analysis systems for engineering drawings must discriminate between text and graphics (mainly lines). This is usually accomplished very well except for some confusion when characters adjoin lines, causing them to be interpreted as graphics or when there are small, isolated graphics symbols that are interpreted as characters.